**1…You are given a string s, and an array of pairs of indices in the string pairs where pairs[i] = [a, b] indicates 2 indices(0-indexed) of the string.You can swap the characters at any pair of indices in the given pairs any number of times. Return the lexicographically smallest string that s can be changed to after using the swaps.**

def smallest\_string\_with\_swaps(s, pairs):

parent = list(range(len(s)))

def find(x):

if parent[x] != x:

parent[x] = find(parent[x])

return parent[x]

for a, b in pairs:

parent[find(a)] = find(b)

groups = {}

for i, char in enumerate(s):

root = find(i)

groups.setdefault(root, []).append(char)

for group in groups.values():

group.sort(reverse=True)

return ''.join(groups[find(i)].pop() for i in range(len(s)))

# Example usage:

s = "dcab"

pairs = [[0, 3], [1, 2], [0, 2]]

print(smallest\_string\_with\_swaps(s, pairs))

**2.Given two strings: s1 and s2 with the same size, check if some permutation of string s1 can break some permutation of string s2 or vice-versa. In other words s2 can break s1 or vice-versa. A string x can break string y (both of size n) if x[i] >= y[i] (in alphabetical order) for all i between 0 and n-1.**

def can\_break(s1, s2):

return all(ord(c1) >= ord(c2) for c1, c2 in zip(sorted(s1), sorted(s2))) or \

all(ord(c2) >= ord(c1) for c1, c2 in zip(sorted(s1), sorted(s2)))

# Example usage:

s1 = "abc"

s2 = "xya"

print(can\_break(s1, s2))  # Output: True

**3. You are given a string s. s[i] is either a lowercase English letter or '?'. For a string t having length m containing only lowercase English letters, we define the function cost(i) for an index i as the number of characters equal to t[i] that appeared before it, i.e. in the range [0, i - 1]. The value of t is the sum of cost(i) for all indices i. For example, for the string t = "aab":**

**cost(0) = 0**

**cost(1) = 1**

**cost(2) = 0**

**Hence, the value of "aab" is 0 + 1 + 0 = 1. Your task is to replace all occurrences of '?' in s with any lowercase English letter so at the value of s is minimized.**

def min\_cost(s):

char\_counts = Counter(c for c in s if c.isalpha())

cost = 0

result = ""

for char in s:

if char == '?':

min\_char = min(char\_counts.keys(), key=char\_counts.get)

result += min\_char

cost += char\_counts[min\_char]

char\_counts[min\_char] -= 1

else:

result += char

cost += char\_counts[char]

return result, cost

s = "aab?c??"

result, cost = min\_cost(s)

print(result, cost)

**4. You are given a string s. Consider performing the following operation until s becomes empty: For every alphabet character from 'a' to 'z', remove the first occurrence of that character in s (if it exists). For example, let initially s = "aabcbbca". We do the following operations: Remove the underlined characters s = "aabcbbca". The resulting string is s = "abbca". Remove the underlined characters s = "abbca". The resulting string is s = "ba". Remove the underlined characters s = "ba". The resulting string is s = "". Return the value of the string s right before applying the last operation. In the example above, answer is "ba".**

def reduce\_string(s):

seen = set()

for char in s:

if char.isalpha() and char not in seen:

seen.add(char)

else:

return s

return ""

s = "aabcbbca"

result = reduce\_string(s)

print(result)

**5. Given an integer array nums, find the subarray with the largest sum, and return its sum.**

**Example 1:**

**Input: nums = [-2,1,-3,4,-1,2,1,-5,4]**

def max\_subarray\_sum(nums):

max\_current = max\_global = nums[0]

for num in nums[1:]:

max\_current = max(num, max\_current + num)

max\_global = max(max\_global, max\_current)

return max\_global

nums = [-2, 1, -3, 4, -1, 2, 1, -5, 4]

print("Maximum subarray sum is:", max\_subarray\_sum(nums))

**6. You are given an integer array nums with no duplicates. A maximum binary tree can be built recursively from nums using the following algorithm: Create a root node whose value is the maximum value in nums. Recursively build the left subtree on the subarray prefix to the left of the maximum value. Recursively build the right subtree on the subarray suffix to the right of the maximum value. Return the maximum binary tree built from nums.**

6->class TreeNode:

def \_init\_(self, val=0, left=None, right=None):

self.val = val

self.left = left

self.right = right

def construct\_maximum\_binary\_tree(nums):

if not nums:

return None

max\_index = nums.index(max(nums))

root = TreeNode(nums[max\_index])

root.left = construct\_maximum\_binary\_tree(nums[:max\_index])

root.right = construct\_maximum\_binary\_tree(nums[max\_index + 1:])

return root

def print\_tree(node):

if not node:

return

print(node.val, end=' ')

print\_tree(node.left)

print\_tree(node.right)

nums = [3, 2, 1, 6, 0, 5]

tree = construct\_maximum\_binary\_tree(nums)

print("Pre-order traversal of the constructed maximum binary tree:")

print\_tree(tree)

**7. Given a circular integer array nums of length n, return the maximum possible sum of a non-empty subarray of nums.A circular array means the end of the array connects to the beginning of the array. Formally, the next element of nums[i] is nums[(i + 1) % n] and the previous element of nums[i] is nums[(i - 1 + n) % n].A subarray may only include each element of the fixed buffer nums at most once. Formally, for a subarray nums[i], nums[i + 1], ..., nums[j], there does not exist i <= k1, k2 <= j with k1 % n == k2 % n.**

def maxCircularSum(a, n):

if (n == 1):

return a[0]

sum = 0

for i in range(n):

sum += a[i]

curr\_max = a[0]

max\_so\_far = a[0]

curr\_min = a[0]

min\_so\_far = a[0]

for i in range(1, n):

curr\_max = max(curr\_max + a[i], a[i])

max\_so\_far = max(max\_so\_far, curr\_max)

curr\_min = min(curr\_min + a[i], a[i])

min\_so\_far = min(min\_so\_far, curr\_min)

if (min\_so\_far == sum):

return max\_so\_far

return max(max\_so\_far, sum - min\_so\_far)

a = [11, 10, -20, 5, -3, -5, 8, -13, 10]

n = len(a)

print("Maximum circular sum is", maxCircularSum(a, n))

**8. You are given an array nums consisting of integers. You are also given a 2D array** queries**, where queries[i] = [posi, xi].For query i, we first set nums[posi] equal to xi, then we calculate the answer to query i which is the maximum sum of a subsequence of nums where no two adjacent elements are selected. Return the sum of the answers to all queries. Since the final answer may be very large, return it modulo 109 + 7. A subsequence is an array that can be derived from another array by deleting some or no elements without changing the order of the remaining elements.**

def smallest\_string\_with\_swaps(s, pairs):

parent = list(range(len(s)))

def find(x):

if parent[x] != x:

parent[x] = find(parent[x])

return parent[x]

for a, b in pairs:

parent[find(a)] = find(b)

groups = {}

for i, char in enumerate(s):

root = find(i)

groups.setdefault(root, []).append(char)

for group in groups.values():

group.sort(reverse=True)

return ''.join(groups[find(i)].pop() for i in range(len(s)))

s = "dcab"

pairs = [[0, 3], [1, 2], [0, 2]]

print(smallest\_string\_with\_swaps(s, pairs))

**9. Given an array of points where points[i] = [xi, yi] represents a point on the X-Y plane and an integer k, return the k closest points to the origin (0, 0).The distance between two points on the X-Y plane is the Euclidean distance (i.e., √(x1 - x2)2 + (y1 - y2)2). You may return the answer in any order. The answer is guaranteed to be unique (except for the order that it is in).**

**10. Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).**

10.def Solution(arr):

n = len(arr)

if n % 2 == 0:

z = n // 2

e = arr[z]

q = arr[z - 1]

ans = (e + q) / 2

return ans

else:

z = n // 2

ans = arr[z]

return ans

if \_name\_ == "\_main\_":

arr1 = [-5, 3, 6, 12, 15]

arr2 = [-12, -10, -6, -3, 4, 10]

arr3 = arr1 + arr2

arr3.sort()

print("Median = ", Solution(arr3))